Introduction to SQL

Class 6

Course Overview

- Introduction to SQL
  - Databases, Tables
  - Classification of SQL – DDL, DML, DCL, TCL
    - DDL – CREATE, ALTER, DROP
    - DML – SELECT, INSERT, UPDATE, DELETE
    - DCL – GRANT, REVOKE
    - TCL – COMMIT, ROLLBACK, SAVEPOINT
  - Data types, Operators
  - Keys – Primary, Foreign, Composite, Unique, Alternate
  - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
  - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
  - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, FOR, LOOP), Stored Procedures,Stored functions
  - Views
  - Indexes, Cursors, Triggers, Events
  - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
  - Tuning SQL queries and optimizing performance
  - SQL Databases vs NoSQL Databases
  - ACID, CAP
  - How SQL databases internally works

FOREIGN KEY

- A foreign key is a column or set of columns in a table that references a primary key in another table.
- It is used to establish a link between data in two tables and ensure referential integrity.
- A foreign key constraint is used to prevent actions that would destroy links between tables.
- It can be implemented on a single column or multiple columns of a table.
- A foreign key can reference either a unique key or a primary key of the referenced table.
- Foreign keys can also be used to create relationships between tables, such as a one-to-many or many-to-many relationship.
- The referenced table is called the parent table, while the table with the foreign key is called the child table.
- A table can have multiple foreign keys that reference different tables.
- You can use the ALTER TABLE and CREATE TABLE statements to add foreign keys to a table.
- Foreign keys can be dropped using the ALTER TABLE or DROP TABLE statement.

COMPOSITE KEY

- A composite key is a combination of two or more columns that together form a primary key for a table.
- It is used to uniquely identify each row in a table.
- Composite keys can be used when a single column is not sufficient to uniquely identify a row.
- It can be used to enforce the integrity of the data in a table by ensuring that there are no duplicate values.
- A composite key constraint is used to prevent actions that would create duplicate values in the composite key columns.
- The columns included in the composite key should be chosen carefully to ensure that they together uniquely identify a row.
- A composite key can be used to join multiple tables together in a relational database.
- You can use the ALTER TABLE or CREATE TABLE statement to add composite key to a table.
- Composite keys are often used as index for a table, which can improve the performance of queries that search for specific rows.
- In MySQL, a composite key can be created by listing multiple columns in the PRIMARY KEY clause of a CREATE TABLE or ALTER TABLE statement.

Alternate Key (Secondary key, non primary key)

- It is a unique key that is not used as the primary key of a table.
- It is a column or set of columns that can be used to uniquely identify a row in a table, but is not used as the main reference for the table.
- Alternate keys are defined using the UNIQUE constraint.
- For example, consider a table called "employees" that has the following columns:
  - id (primary key)
  - email (unique)
  - social_security_number (unique)
  - Here email and social_security_number are alternate keys

Customer table

```
CREATE TABLE customer (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL,
    city VARCHAR(50) NOT NULL,
    state VARCHAR(50) NOT NULL,
    zip_code VARCHAR(10) NOT NULL,
    phone VARCHAR(15) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Employee Table

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    salary DECIMAL(10, 2) NOT NULL,
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

Employee table

```
CREATE TABLE employee (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    age INT NOT NULL CHECK (age > 18),
    salary DECIMAL(10,2) NOT NULL,
    department VARCHAR(50) NOT NULL
);
```

Tweets

```
CREATE TABLE tweets (
    id BIGINT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    username VARCHAR(255) NOT NULL,
    text VARCHAR(280) NOT NULL,
    created_at DATETIME NOT NULL,
    location VARCHAR(255),
    lang VARCHAR(10) NOT NULL,
    favorite_count INT,
    retweet_count INT,
    UNIQUE (id,user_id)
);

INSERT INTO tweets (id, user_id, username, text, created_at, location, lang, favorite_count, retweet_count)
VALUES (1234567891, 9876543211, 'twitteruser1', 'This is another tweet!', '2022-01-01 12:00:00', 'Los Angeles, CA', 'en', 200, 150),
(1234567892, 9876543212, 'twitteruser2', 'This is yet another tweet!', '2022-01-01 12:00:00', 'Chicago, IL', 'en', 150, 100);
```

Images

```
CREATE TABLE images (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    image_path VARCHAR(255) NOT NULL,
    created_at DATETIME NOT NULL,
    UNIQUE (name,image_path)
);
```

Local Images

```
CREATE TABLE images (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    image BLOB NOT NULL,
    created_at DATETIME NOT NULL,
    UNIQUE (name)
);

INSERT INTO images (name, description, image, created_at)
VALUES ('image1', 'This is an image', LOAD_FILE('path/to/image1.jpg'), NOW());


INSERT INTO images (name, description, image, created_at)
VALUES ('image2', 'This is another image', LOAD_FILE('path/to/image2.jpg'), NOW()),
('image3', 'This is yet another image', LOAD_FILE('path/to/image3.jpg'), NOW());
```

Orders table

```
CREATE TABLE orders (
    id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    CHECK (price * quantity > 100)
);
```

Orders table

```
CREATE TABLE orders (
   id INT,
   customer_id INT,
   product_id INT,
   quantity INT NOT NULL,
   price DECIMAL(10,2) NOT NULL,
   PRIMARY KEY (id, customer_id),
   FOREIGN KEY (customer_id) REFERENCES customers(id),
   FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Country Table

```
CREATE TABLE country (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    continent ENUM('Africa', 'Antarctica', 'Asia', 'Europe', 'North
America', 'Australia', 'South America') NOT NULL,
    PRIMARY KEY (id)
);
```

Products with ENUM

```
CREATE TABLE products (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    status ENUM('in stock', 'out of stock', 'discontinued') NOT NULL,
    PRIMARY KEY (id)
);
```

Employee with enum

```
CREATE TABLE employees (
   id INT NOT NULL AUTO_INCREMENT,
   name VARCHAR(255) NOT NULL,
   position ENUM('manager', 'supervisor', 'employee') NOT NULL,
   PRIMARY KEY (id)
);
```

Orders with enum

```
CREATE TABLE orders (
    id INT NOT NULL AUTO_INCREMENT,
    order_date DATE NOT NULL,
    payment_method ENUM('credit card', 'debit card', 'cash') NOT NULL,
    PRIMARY KEY (id)
);
```

Alter enums later

```
ALTER TABLE orders
MODIFY payment_method ENUM('credit card', 'debit card', 'cash', 'online wallet’);
```

```
ALTER TABLE orders
MODIFY payment_method ENUM_ADD('online wallet');
```

Orders

```
CREATE TABLE orders (
    id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    CHECK (price * quantity > 100) CONSTRAINT min_order_value,
    FOREIGN KEY (customer_id) REFERENCES customers(id) ON DELETE
CASCADE,
    FOREIGN KEY (product_id) REFERENCES products(id) ON DELETE CASCADE
);
```

Books and Authors - Foreign Key Example

```
CREATE TABLE books (
  book_id INT PRIMARY KEY,
  title VARCHAR(255),
  ISBN VARCHAR(255),
  publication_date DATE,
  author_id INT,
  FOREIGN KEY (author_id) REFERENCES authors(author_id)
);

CREATE TABLE authors (
  author_id INT PRIMARY KEY,
  name VARCHAR(255),
  nationality VARCHAR(255),
  birthdate DATE
);
```

```
INSERT INTO books (book_id, title, ISBN, publication_date,
author_id)
VALUES (1, 'To Kill a Mockingbird', '0-449-00295-5', '1960-07-
11', 1),
       (2, 'The Great Gatsby', '0-679-60167-3', '1925-04-10', 2),
       (3, 'Moby-Dick; or, The Whale', '0-385-13560-2', '1851-10-
18', 3);

INSERT INTO authors (author_id, name, nationality, birthdate)
VALUES (1, 'Harper Lee', 'American', '1926-04-28'),
       (2, 'F. Scott Fitzgerald', 'American', '1896-09-24'),
       (3, 'Herman Melville', 'American', '1819-08-01');
```

Domain/Entity/Referential Integrity Constraints

- **Domain integrity** constraints are used to ensure that the values entered into a database meet certain rules or conditions, such as data type, size, or format.
  - For example, a domain integrity constraint might require that a date field only contain valid dates.

- **Entity integrity** constraints are used to ensure that each row in a database table is unique, typically by enforcing a primary key constraint. This means that no two rows can have the same primary key value.

- **Referential integrity** constraints are used to ensure that relationships between tables in a database are maintained. These constraints are typically enforced using foreign key constraints, which link a column in one table to the primary key of another table.
  - For example, a referential integrity constraint might ensure that a value in the "customer_id" column of an "orders" table matches a value in the "customer_id" column of a "customers" table.

Alter table examples

- ALTER TABLE orders ADD COLUMN shipping_address VARCHAR(255);
- ALTER TABLE customers MODIFY email VARCHAR(255) NOT NULL;
- ALTER TABLE products CHANGE name product_name VARCHAR(255);
- ALTER TABLE orders DROP COLUMN shipping_address;
- ALTER TABLE orders ADD PRIMARY KEY (id);
- ALTER TABLE orders ADD FOREIGN KEY (customer_id) REFERENCES customers(id);
- ALTER TABLE orders ADD CHECK (price * quantity > 100);
- ALTER TABLE orders DROP FOREIGN KEY FK_orders_customer_id;
- ALTER TABLE orders RENAME TO customer_orders;
- ALTER TABLE customer ADD UNIQUE (email);
- ALTER TABLE orders MODIFY price DECIMAL(10,4);

ADD/RENAME/DELETE Columns

Adding a single column

```
ALTER TABLE table_name
   ADD COLUMN column_name column_definition [FIRST|AFTER existing_column];
```

Adding multiple columns

```
ALTER TABLE table_name
   ADD COLUMN column_name1 column_definition [FIRST|AFTER existing_column],
   ADD COLUMN column_name2 column_definition [FIRST|AFTER existing_column];
```

ADD/RENAME/DELETE Columns

Renaming a column name using CHANGE

```
ALTER TABLE table_name
    CHANGE COLUMN old_column_name new_column_name column_definition [FIRST|AFTER existing_colu
mn];
```

The column definition is compulsory even if we are not changing it

Renaming a column name using RENAME

```
ALTER TABLE table_name
RENAME COLUMN old_column_name TO new_column_name;
```

Renaming multiple columns

```
ALTER TABLE table_name
CHANGE old_column_name1 new_column_name1 Data Type,
CHANGE old_column_name2 new_column_name2 Data Type,
```

```
ALTER TABLE table_name
RENAME COLUMN old_column_name1 TO new_column_name1,
RENAME COLUMN old_column_name2 TO new_column_name2,
```

ADD/RENAME/DELETE Columns

Dropping a column

ALTER TABLE table_name **DROP COLUMN** column_n
ame;

Dropping multiple columns

ALTER TABLE table_name
DROP COLUMN column_1,
DROP COLUMN column_2,

Points to remember while deleting a column

- When we remove columns from a table, it will affect all associated objects such as triggers, stored procedures, and views. Suppose we delete a column that is referencing in the trigger. After removing the column, the trigger becomes invalid.
- The dropped column depends on other applications code, must also be changed, which takes time and effort.
- When we remove a column from the large table, it will affect the database's performance during removal time.

TRUNCATE Table

- The TRUNCATE TABLE statement in MySQL is used to remove all data from a table and reset the auto-increment value to its starting value (usually 0 or 1).
- TRUNCATE TABLE is faster than DELETE statement, because it doesn't generate any undo logs and doesn't fire any triggers.
- TRUNCATE TABLE is not a DML statement, it is a DDL statement
- Truncating a table also resets the AUTO_INCREMENT counter to its starting value.
- Truncate table can't be rollback, unlike DELETE statement
- TRUNCATE TABLE statement is used to delete all the rows from the table and free the space containing the data.
- TRUNCATE TABLE statement is used to delete all the rows from a table and reset the auto-increment value.
- TRUNCATE TABLE can't be used on a table with foreign key references.
- TRUNCATE TABLE statement is faster than DELETE statement as it does not generate any undo logs and doesn't fire any triggers.
- TRUNCATE TABLE statement is used to empty a table quickly and efficiently.

Difference between DELETE / DROP / TRUNCATE

	DELETE Command	DROP Command	TRUNCATE Command
Language	The DELETE command is Data Manipulation Language Command.	The DROP command is Data Definition Language Command.	The TRUNCATE command is a Data Definition Language Command.
Use	The DELETE command deletes one or more existing records from the table in the database.	The DROP Command drops the complete table from the database.	The TRUNCATE Command deletes all the rows from the existing table, leaving the row with the column names.
Transition	We can restore any deleted row or multiple rows from the database using the ROLLBACK command.	We cannot get the complete table deleted from the database using the ROLLBACK command.	We cannot restore all the deleted rows from the database using the ROLLBACK command.
Memory Space	The DELETE command does not free the allocated space of the table from memory.	The DROP command removes the space allocated for the table from memory.	The TRUNCATE command does not free the space allocated for the table from memory.
Performance Speed	The DELETE command performs slower than the DROP command and TRUNCATE command as it deletes one or more rows based on a specific condition.	The DROP Command has faster performance than DELETE Command but not as compared to the Truncate Command because the DROP command deletes the table from the database after deleting the rows.	The TRUNCATE command works faster than the DROP command and DELETE command because it deletes all the records from the table without any condition.
Integrity Constraints	The Integrity Constraints remain the same in the DELETE command.	The Integrity Constraints get removed for the DROP command.	The Integrity Constraints will not get removed from the TRUNCATE command.
Permission	DELETE permission is required to delete the rows of the table.	We need ALTER permission on the schema to which the table belongs and CONTROL permission on the table to use the DROP command.	We need table ALTER permission to use the TRUNCATE command.
Syntax	DELETE FROM table_name WHERE condition;	DROP TABLE table_name;	TRUNCATE TABLE table_name;

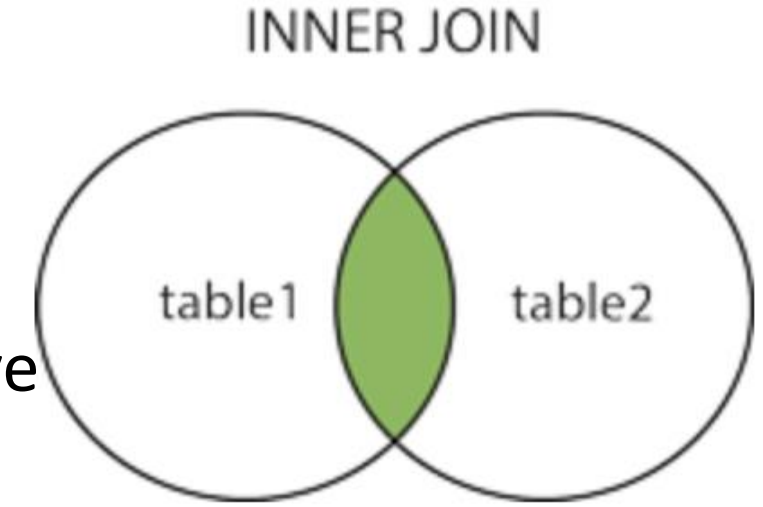
JOINS – Why do we need joins

- Combining data from multiple tables into one result set.
- Retrieving data from multiple tables based on a common column between them.
- Merging data from tables with one-to-one, one-to-many, or many-to-many relationships.
- Consolidating data from separate tables into a single view.
- Retrieving additional data not present in one table by joining it with another.

SQL Join

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Different types of joins :
  - ❑ (INNER) JOIN: Returns records that have matching values in both tables.
  - ❑ LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table.
  - ❑ RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table.
  - ❑ FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table.

SQL Inner Join



- The INNER JOIN keyword selects records that have matching values in both tables.
- This is also the default type of join
- SYNTAX :

```
 SELECT column_name(s)
 FROM table1
 INNER JOIN table2
ON table1.column_name = table2.column_name
INNER JOIN table3
ON table2.column_name = table3.column_name;
```

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)

Thank you